MICROSOFT®

TECHNICAL JOURNAL

Microsoft Confidential **Editor, Ross Garmoe**

March 14, 1984
Volume I

"The goal of modern engineering is a clean elegant design with a minimum of parts."

R. Goldberg

Editor's Comments - Ross Garmoe

Since its inception, Microsoft has enjoyed phenomenal success. Our products are well accepted and in many cases set the standard. Sales and personnel have doubled every year. This success is enjoyable but it does impact all of our jobs. The increase in development staff calls for greater effort be dedicated to communications with other groups and even with people within our own groups. To continue our success, our products must become increasingly more sophisticated and the various products must work well together. The demands of the marketplace require that we increase our productivity and the quality of the end product. To be sure that the products we deliver perform correctly and solve the right problems, we need to be able to rapidly prototype and change them as required. Tools and techniques developed in one group need to be shared across the company so work is not duplicated and the productivity of everybody is improved.

There are many things that can be done to improve personal and company productivity but the major areas where we need to expand our effort are:

- o improved development environment
- o improved development tools and procedures
- o thorough and professional testing techniques
- o standardized development tools and approaches
- o shared code and designs
- o begin building a base of Microsoft Software technology instead of starting every project from scratch
- o improved communication of techniques and solutions

To provide a vehicle for exchange of information, we are publishing the Microsoft Technical Journal on a monthly basis. Although we are calling it a technical journal, it will have much the flavor of a newsletter in that it will be informal in style and content. Articles will be published as received with only a quick review for grammar and spelling errors.

One of the major points of discussion after the decision was made to publish the journal was about the content. The original plan was to not restrict the distribution and publish only "safe" articles containing only information that was already known outside of Microsoft. Several people immediately objected, stating that the journal would contain only articles of historical interest and they would be prohibited from writing about what they considered interesting and important. We recognized that this was a valid criticism and would reduce the usefulness of the journal. Please note that the journal is considered a Microsoft confidential document and its contents should be protected as is any other confidential document. If information published here is made known outside the company, our policy will have to be reviewed. If we start restricting what can be published, then the value of the journal will be reduced and we will all lose.

The journal will be published on the second Wednesday of every month. Articles to be included in an issue should be sent to me by the first Wednesday of the month via email or internal mail. Email articles should be sent to "rossg" and should be clearly marked for submission to the journal. If you send a hard copy through the internal mail system, please allow

additional time and be sure that it is a clean copy. The typing of the journal will be done by Word Processing and hard-to-read copy will increase their workload and the chances for errors.

The journal will also be used as a means of distributing manual pages for the various tools developed or purchased for in-house use. Manual pages can be submitted either in hard copy format or as input to one of the text processing systems. In general, we prefer the text processor input because that makes it easier for us to regenerate the documentation when the need arises. Manual pages can be submitted at any time, and copies will be distributed with the next journal issue.

The Microsoft Technical Journal and documentation will be printed on 3-hole punched paper so that you can keep the material on hand for easy reference. You have also received a notebook with printed dividers for the journal, Xenix, MS-DOS, TOPS-20 and general documentation manual pages. These notebooks have both front and spine labels identifying the book as MS Internal Tools. Please be careful to not let these notebooks get lost or stolen. Each new technical person will be provided with an updated notebook of past journals and documentation when they join the company.

Everybody is encouraged to submit articles on technical topics. These can be either about current Microsoft projects or topics in which you have a personal interest. Please do not be bashful about submitting articles. The journal will be only as good as you make it. We are also very interested in any comments or suggestions that you have for improving the journal.

Tools Environment - Bill Gates

Producing software efficiently is critical to Microsoft. Efficient production allows us to: (a) use small groups that work as a team to create significant products; (b) respond to new hardware features and customer needs ahead of others; (c) spend more time designing and tuning for speed and space, and (d) enjoy programming more. Although we have several great development tools, overall our development efficiency relies primarily on our excellent programming abilities.

Despite a strong belief in tools, on the balance we have only average development environments (some better than average, some worse). 1984 is our chance to share ideas and invest in making our tools the best. Obvious utilities like downloaders, symbolic debuggers and profiling packages can be created and improved with immediate significant impact.

We used DEC-20 timesharing to develop 8-bit software because it provided shared files, symbolic debugging, daily backup, large disks, and a full set of utilities. Today microcomputers can be better development environments and we have decided to move away from the DEC-20 except for ongoing work on 8-bit products. The lack of a remote file system facility on MS-DOS and Xenix has prevented us from making an immediate shift. Ironically, we provide the world with resident development tools but we haven't made use of these tools in-house. Although our first reaction to use of our own tools is sometimes to

wonder how other people get by, this is a great opportunity to strengthen the development tools we sell. Selling development tools is a large business for us and having good ones builds a reputation with a group that influences a lot of end users. Our assembler and linker can be made much faster and given more features now that we will be using them and we have a focus on tools. Bringing together the benefits of Cmerge and OS, putting the key Xenix tools on MS-DOS, passing local symbols to the debuggers, etc. are projects that will get done if we get the input that these are the things that will help. I am sure that there are other ideas to improve development environments of equal or greater importance. It is up to everyone to get involved in improving our efficiency.

The next key step is the choice of our single user workstation for development. Although we may not be able to be all things to all people, the committee evaluting the choices, Gordon Letwin and Charles Simonyi, will do a better job if all ideas are pooled.

Development Machines - Gordon Letwin

Microsoft is currently finalizing the specifics of our next generation development environment. We are evaluating commercially available systems and will finalize our choice shortly. The specification "must" list includes:

1) processor-on-a-desktop design. A "rather powerful" processor, with hardware protection and relocation

2) a hi-res bit-mapped screen

3) a high-speed network interconnect

4) minimal memory of one megabyte; easily expandable to two meg with potential for more

5) most systems will have little or no local disk storage

6) the key software (including the OS) must be maintained and enhanced locally; we cannot rely on others for critical software.

A "typical" system that we're considering is the Sun 2/120:

10 mhz 68010 (a 68000 with virtual memory capability)
1024 x 1025 hi-res screen
optical mouse
ethernet
multibus

Although packed more elegantly, this is virtually the same hardware that makes up each of our 68K boxes. The major difference being that

1) you'll have the box to yourself, and

2) you'll be more closely coupled to all the other machines via the network and MS-NET. Our current UFTP/UVTP mechanisms are NOT "closely coupled".

These systems will support a variety of window-oriented development tools, with an underlying tools-based operating system which has a superset of XENIX's functionality. The UNIX tools will also be available.

Please note that we will not be trying to force programmers to exclusively use menus and mice. The window environment is very powerful and supports interfaces that range from the existing "character oriented glass tty" to "idiot-proof" menu/mouse packages. Most programmers will probably work in a range of environments, using "efficient and cryptic" interfaces for things they do often, using the more helpful interfaces for things they do less frequently.

/usr/tools - Ross Garmoe

With the hope and expectation that all of this talk about tools will encourage everybody to submit their favorite programs, we have established the directory /usr/tools on all of the Xenix systems. This directory will contain useful programs that are not part of the standard Xenix system but are important enough for everybody to have access to. The rules for submission to /usr/tools are relatively simple:

a) All submissions must be in program source form. We must be able to create the program for Xenix on all of the different systems. "Make" files will be greatly appreciated.

b) Manual pages must be provided in troff source form. This will allow us to generate the manual pages without having to bother you for a new copy.

The executable programs will be placed on all of the Xenix systems. The program and documentation source will be stored in /usr/tools on the machine sprocket in SCCS format.

Notes from the Library - Natalie Yount

The charter of the Microsoft Technical Library is to be a working reference collection in support of the technical groups. In addition to organizing, we are trying to develop the collection in the technical areas. You can help by suggesting titles of books, journals, technical reports, conferences, etc. that are of interest to you and by providing feedback on the library collection and service.

Listed below are some new additions to the Microsoft Library. If you would like to see any of these titles, send mail to library. (NOTE: Do not be surprised if these items are not all on the shelf - unless they are strictly reference material - they are no good to Microsoft if they are not being used! Ask librarian for assistance.)

LIBRARY ADDITIONS (2/27/84)

ALGORITHMIC PROGRAM DEBUGGING. E. Shapiro. Mit Press, 1982.

ALGORITHMS + DATA STRUCTURES = PROGRAMS. Niklaus Wirth. Prentice-Hall, 1976.

COMPUTER IMAGES: STATE OF THE ART. J. Deken, 1983.

DATA COMPRESSION: TECHNIQUES AND APPLICATIONS HARDWARE AND SOFTWARE CONSIDERATION. Gilbert Held. John Wiley, 1983.

OPERATING SYSTEMS. S. Madnick. McGraw-Hill, 1974.

DEVELOPING MANAGERIAL SKILLS IN ENGINEERS AND SCIENTISTS. M. Badawy, Van Nostrand, 1982.

ELEMENTARY PASCAL. Henry Ledgard, 1982.

8088 ASSEMBLER LANGUAGE PROGRAMMING: IBM PC. David Willen. Howard Sams, 1983.

HISTORY OF PROGRAMMING LANGUAGES. R. Wexelblat. Academic Press, 1981.

THE IBM COBOL ENVIRONMENT. Robert Grauer. Prentice-Hall, 1984.

INTRODUCTION TO DATABASE SYSTEMS. C. Date. Addison-Welsey, 1983. Vol. 2.

MS-DOS USERS GUIDE. Chris DeVoney. Que, 1984.

MICROCOMPUTER OPERATING SYSTEMS. Mark Dahmke. Byte 1982

A PRACTICAL GUIDE TO THE UNIX SYSTEM. Mark Sobell. Benjamin, 1984.

SCREEN DESIGN STRATEGIES FOR COMPUTER ASSISTED INSTRUCTION. J. Heines. Digital Press, 1984.

Column C - Ralph Ryan

This column will present topics on the C language and on the new Microsoft C compiler. This issue will describe the memory models and options available with the new compiler. In future issues there will be columns on portability, the ANSI standards effort, and explanations of subtler aspects of the C language. Suggestions are always welcome.

This column describes the features of the Cmerge C compiler generating code for the 8086/186/286. The options described here are only meaningful to x86 machines.

CONFIGURATION STRING

The command line of msc will be processed for a configuration string of the form "-M<string>". The values that can be used in the <string> are:

```
s   - small model
e   - hybrid model (near and far keywords enabled)
m   - middle model
l   - long model
h   - huge model
2   - generate 286 instructions
b   - use old Xenix word order
t#  - the number ( # ) represents the allocation threshold for long model.
      See the description of long model below.
```

SEGMENT NAMING

Each of the memory models described below has default names for the segments. These names can be changed using the following switches:

```
-NT<string>  - rename the text segment for this compiland
-ND<string>  - rename this compiland's default data segment
-NM<string>  - rename the module name (default is compiland name)
```

MEMORY MODELS

Two key terms:

"near" objects have a 16-bit address (relative to DS)
"far" objects have a full 32-bit segmented address

Small model:

One data segment and one code segment
DS=SS (default name is _DATA) and CS (default name is _TEXT)
This is the default mode. The configuration string can be left off, or entered as -Ms.

Hybrid model;

One data segment and one code segment
DS=SS (default name is _DATA) and CS (default name is _TEXT)
This is the same as small model, except that the near and far keywords are enabled. Different code is generated for structure copies.

Middle model:

One data segment and multiple code segments. All calls or returns are far calls or returns.
DS=SS (default name is _DATA) CS is separate per module (default name is <module> _TEXT). The configuration string is -Mm.

Large model:

Multiple data segments and multiple code segments. No data item may be larger that one segment (64K). Items smaller than the data threshold (default is 128) are targeted to the default data segment (_DATA). All other data items are allocated their own segment. All calls and returns are far. The configuration string is -Ml. If the threshold is to be changed to 64 (for example), the configuration string is -Mlt64.

Huge model:

Same as long model, but removes the object size restriction. There are further constraints that will be discussed in a separate memo.

Model	Data Pointer	Code Pointer	Integer Size
small	16	16	16
middle	16	32	16
large	32	32	16
huge	32	32	16

NEAR AND FAR EXTENSIONS TO C

One of the limitations of the memory model structure is that all items change size at once. Thus, in a middle model program all procedures are "far" and all data is "near". For efficient programs it is desirable to create "mixed" models -- in particular, small model programs that can access a few "far" data items. The Cmerge C compiler has been enhanced to include the keywords "near" and "far" (enabled by using an 'e' in the configuration string). These words bind immediately to the right, and modify the declaration. In the examples that follow, it is assumed that a small model program is being compiled. A key concept is the distinction between the address of an object, and the contents of that object.

Declaration	Address of Object	Contents of Object
char c;	near (16 bits)	8 bits (data)
char far d;	far (32 bits)	8 bits (data)
char *p;	near (16 bits)	16 bits (near pointer)
char far *q;	near (16 bits)	32 bits (far pointer)
char * far r;	far (32 bits)	16 bits (near pointer)[1]
char far * far s;	far (32 bits)	32 bits (far pointer)[2]
int foo();	near (16 bits)	function returning 16 bits
int far foo();	far (32 bits)	function returning 16 bits[3]

NOTES: 1) Note that the example is meaningless, and exists only for syntactic completeness.

2) Long model could be synthesized in this fashion.

3) This example leads to trouble in most environments. The far call changes the CS register, and makes runtime support unavailable. Again, the example is included for syntactic completeness. A better example is from a middle model compilation:

```
int near foo();
```

does a near call in an otherwise far (calling) program. In the absence of inter-compiland type checking, such features should be used with great care.

Xenix to MS-DOS Tools Port - Rueben Borman

For the past several months, I have been porting as many Xenix utilities as possible to MS-DOS.

There are three main reasons for this project: First, MS-DOS lacks a standard set of tools which provide the power and functionality of the Xenix utilities. Second, there is a transition problem for users who must go back and forth between Xenix and MS-DOS; the existence of many of the same tools in both environments will ease the transition. Finally, it is important to develop a C language environment in MS-DOS which allows easy transporting of C programs developed on either Xenix for MS-DOS from one system to the other.

The porting of the Xenix tools required such an environment to be developed on MS-DOS and provided useful information on the limits and problems of the goal of C program portability across the two systems. These problems included operating system issues such as CRLF vs. LF, multitasking, and C language issues such as signed characters. The issues have been successfully resolved in the case of the utilities that were ported.

So far, the following Xenix utilities of interest are available on MS-DOS:

ar	cmp	diff	ed	fgrep
grep	mkstr	more	od	split
tail	tr	vi		

Users, who are familiar with Unix systems, will particularly appreciate diff, grep, more, and od, which are respectively a differential file comparator, regular expression searcher, sophisticated file viewer, and byte or word dumper.

Some people have questioned why I ported vi since it is huge (119K) and Z is available. It is important to remember that programmers spend a lot of time in editors and get used to a particular one. Having vi available on MS-DOS, even if it is big and somewhat slow, is quite valuable for a vi user on Xenix who has to spend time working on MS-DOS.

Initially I used Lattice C to port the tools. A set of Unix system calls and runtime-library functions had to be ported or simulated, for example, ioctl, stat, fstat, and signal. This was accomplished with varying degrees of success with the help of Mark Zbikowski's TOOLS runtime library. Currently I am moving over to the C Merge compiler and runtime library which are considerably superior to Lattice for porting programs written on Xenix.

The goal remains to port MS-DOS everything from Xenix that is possible to be ported. The list of tools next under consideration includes tar, make, sed, lint, lex, and awk.

The utilities are available from Reuben Borman, Ext. 2250. They can be copied from the diskettes on by door, or you can send a blank diskette (two if you want vi) and the serial number of your PC). Xenix is a licensed version of Unix which is a copyrighted product of ATT. Your machine must have a *binary* license for Unix to receive a copy of these utilities. If you will send me the serial number of your machine, we will handle the licensing process.